Particle Swarm Inspired Evolutionary Algorithm (PS-EA) for Multiobjective Optimization Problems

Dipti Srinivasan and Tian Hou, Seow
10 Kent Ridge Crescent, Singapore 119260
Department of Electrical & Computer Engineering
National University of Singapore
dipti@nus.edu.sg, g0200633@nus.edu.sg

Abstract- This paper describes Particle Swarm Inspired Evolutionary Algorithm (PS-EA), which is a hybridized Evolutionary Algorithm (EA) combining the concepts of EA and Particle Swarm Theory. PS-EA is developed in aim to extend PSO algorithm to effectively search in multiconstrained solution spaces, due to the constaints rigidly imposed by the PSO equations. To overcome the constraints, PS-EA replaces the PSO equations completely with a Self-Updating Mechanism (SUM), which emulates the workings of the equations. A comparison is performed between PS-EA with Genetic Algorithm (GA) and PSO and it is found that PS-EA provides an advantage over typical GA and PSO for complex multi-modal functions like Rosenbrock, Schwefel and Rastrigrin functions. An application of PS-EA to minimize the classic Fonseca 2-objective functions is also described to illustrate the feasiblility of PS-EA as a multiobjective search algorithm.

1 Introduction

Population based stochastic search algorithms have been very popular in the recent years in the research arena of computational intelligence. Some well established search algorithms such as Genetic Algorithm (GA) [1-3], Evolutionary Strategies (ES) [4], Evolutionary Programming (EP) [5] and Artificial Immune Systems (AIS) [6], have been successfully implemented to solve simple problems like functions optimization to complex real world problems like scheduling [7-9] and complex network routing problems [3].

Swarm intelligence has become a research interest to many research scientists of related fields in recent years. The main algorithm for swarm intelligence is Particle Swarm Optimization (PSO) [10-14], which is inspired by the paradigm of birds flocking. PSO is successfully implemented in various optimization problems like weight training in Neural Networks [12] and functions optimization [10,11,13,14]. It is very popular due to its simplicity in its implementation, as a few parameters are needed to be tuned. It is computational cheap in the updating of the individuals per iteration, as the core updating mechanism in the algorithm relies only on two simple PSO self-updating equations, as compared to using mutation or crossover operation in typical Evolutionary Algorithm (EA), which requires a substantial computation cost to perform decision making, like which individual shall go for crossover or mutation process.

PSO searches for solution in the solution space differently from a typical EA. An EA iteratively searches for several good individuals in the population, and try to make the population to emulate the best solutions found in that generation through crossover operation, while the mutation operation tries to introduce diversity to the population. The problem of premature convergence occurs often when all individuals in the solutions become very similar to each other. This results the population to be stuck in local optima, if the initial best individual as found by the EA is very near to a local optima. In the workings of PSO, it maintains a memory to store the elite individuals of the best global individual (*gbest*) found, as well as the best solutions as found by each individual (*pbest*). Each individual in the population will try to emulate the *gbest* and *pbest* solutions in the memory through updating by the PSO equations. The random element in the PSO equations introduces diversity around the elite individuals found.

However, even though PSO is a good and fast search algorithm, it has its limitations when solving real world problems. The two PSO equations, which are in the mathematical format, restrict additional heuristics related to the real-world problem to be incorporated in the algorithm, while in the case of EA, heuristics can be easily incorporated in the population generator and mutation operator to prevent wrong updates to the individuals to infeasible solutions. Therefore, PSO will not perform well in its search in complex multi-constrained solution spaces, which are the case for many complex real world problems like scheduling. To overcome the limitations of PSO, this paper proposes a hybridized evolutionary algorithm, which allows flexible incorporations of the real world heuristics into the algorithm, while retaining the workings of PSO.

This paper describes *Particle Swarm Inspired Evolutionary Algorithm* or PS-EA, which is a hybrid model of EA and PSO. PS-EA is compared with PSO and GA on five numerical optimization tasks that are commonly used for benchmarking purposes of optimization algorithms. The results show the advantage of PS-EA over GA and PSO in the optimization of complex functions like Rosenbrock, Schwefel and

Rastrigrin functions. An application of PS-EA to minimize the classic Fonseca 2-objective functions is also described to illustrate the feasiblility of PS-EA as a multi-objective search algorithm.

2 Workings of PS-EA

Particle Swarm Inspired Evolutionary Algorithm (PS-EA) is a hybridized algorithm combining concepts of PSO and EA. The main module of PS-EA is the Self-Updating Mechanism (SUM), which makes use of the Inheritance Probability Tree (PIT) to do the updating operation of each individual in the population. A Dynamic Inheritance Probability Adjuster (DIPA) is incorporated in SUM to dynamically adjust the inheritance probabilities in PIT based on the convergence rate or status of the algorithm in a particular iteration. In this section, the flow of PS-EA and the detailed workings of SUM will be discussed.

2.1 Flow of PS-EA

The general flow of PS-EA is shown as follows:

i) Initialization of initial swarm of particles
ii) Evaluation of particles
iii) Identification of elite particles and save in memory
iv) Undergo Self Updating Mechanism (SUM)
v) Evaluation of particles
vi) Update the elite particles to memory
vii) Repeat (iii)-(vi) until stopping criteria are met

It is similar to most population based stochastic search algorithms, except that it has a memory to store the elite particles or individuals, and there is no reproduction of offspring. All particles in the swarm or population will undergo modifications by SUM. The elite particles include *gbest*, *popbest* and *pbest* particles. It is noted that *popbest* is included as one of the elite particles. The *popbest* particle is the best particle in the current swarm or population. More details will be discussed in the sub-section on SUM.

2.2 Self-Updating Mechanism (SUM)

The Self-Updating Mechanism (SUM) is derived from the concepts of PSO. It functions as an emulator of the PSO self-updating equations as follows:

$$v' = v + c1.U(0,1).\{e1-x\}+c2.U(0,2).\{e2-x\} \quad (1)$$

$$x' = x + v' \quad (2)$$

where v is the velocity vector, $c1$ and $c2$ are the learning factors, $U(0,1)$ is the number generator that produces a number between 0 and 1 based on uniform distribution, $e1$ is *gbest* particle, $e2$ is the *pbest* particle of particle x, and x is the *present* considered particle.

The derivation of SUM, the operation of SUM and the working of DIPA to dynamically adjust the inheritance probabilities of PIT will be described in the following sub sections.

2.2.1 Derivation of SUM

If we analyze the PSO equations as in (1) and (2), we can deduce the following possible results:

- x becomes $e1$ or *gbest* particle
- x becomes $e2$ or *pbest* particle of x
- x remains as it is.
- x is assigned a value near $e1$ or $e2$.

From the analysis of the equations, it is possible to use the operators of EA to emulate the workings of PSO equations. Replacing the PSO equations, we introduce a probability inheritance tree (PIT) as illustrated in Fig. 1.

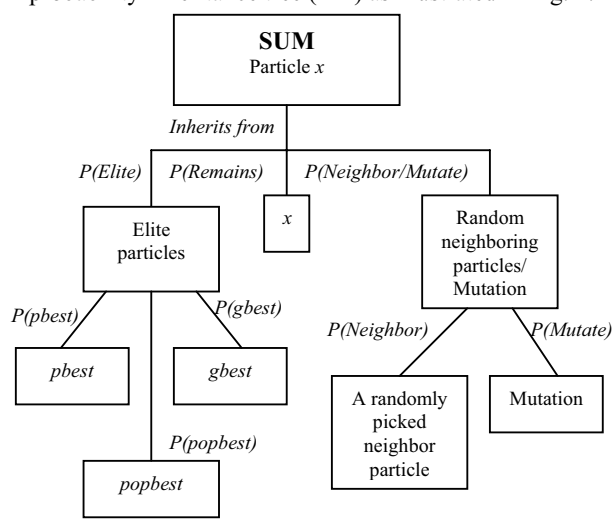

Fig. 1. Probability Inheritance Tree (PIT)

Fig. 1 shows the probability inheritance tree of SUM. The end branches at the bottom of the tree show all the possible results that a parameter value of a particle x can be updated. Particle x can inherit the parameter values from the elite particles or any random neighboring particles, undergoes mutation operation or retains its original value. The mutation operation in SUM emulates the random element in the PSO equation as in (1). An additional elite particle *popbest*, which is the best particle of a current swarm, is introduced to SUM for faster convergence. To introduce more diversity in the swarm of particles, we allow *present[]* to inherit parameter values of a randomly selected neighbor particle in the current swarm.

2.2.2 Operations of SUM

The SUM process can be illustrated by considering the updating of the first parameter of a *particle k.* The first parameter value of *particle k* undergoes the SUM. The parameter has the probability *P(Elite)* to inherit the value of one of the elite particles, probability *P(Remains)* to retain its original value and probability *P(Neighbor/Mutation)* to inherit the value from one of its neighboring particles or undergoes a mutation operation. If it chooses to inherit from one of the elite particles, it still has to choose whether it will inherit from the global best particle *gbest*, the current population best particle *popbest*, or the best particle that *particle k* has in its

memory *pbest[k]*. The probabilities that it will choose to inherit from *gbest*, *popbest* and *pbest[k]* are *P(gbest), P(popbest)* and *P(pbest)* respectively. Similarly, if it chooses to inherit from a neighbor particle or undergoes mutation operation, it will again need to choose between the both options. The probabilities that it will choose to inherit from a randomly selected neighbor particle in the current population and undergoes mutation operation are *P(Neighbor)* and *P(Mutate)* respectively.

By introducing the PIT in SUM, the number of parameters has increased by eight, which are the eight inheritance probabilities. It is therefore a problem for algorithm designer to determine the correct values for the inheritance probabilities, where the total parameters to be set has become nine, considering the eight inheritance probabilities and the population size. Therefore, Dynamic Inheritance Probability Adjuster (DIPA) is proposed to take care in the setting of inheritance probabilities, reducing to setting of only one parameter, which is the population size.

2.2.3 Dynamic Inheritance Probability Adjuster (DIPA)

The dynamic inheritance probability adjuster (DIPA) aims to solve the problem of setting the correct set of inheritance probabilities of the SUM. The DIPA gets feedback on the the convergence status from the convergence rate feedback mechanism, in order to detect whether the algorithm is converging well and whether the search is stuck in some local optima. With the convergence status as feedback from the mechanism, DIPA will adjust the inheritance probability reacting to the convergence status that it gets.

There is this simple feedback mechanism to indicate the convergence status of the algorithm, known as the convergence rate feedback mechanism. For each iteration, the cost of the *gbest* particle will be logged. The cost of the *gbest* particle can indicate whether the algorithm is converging or not. For every even-numbered iteration, the feedback mechanism will calculate the difference of the costs of the *gbest* particle in the odd-numbered and even-numbered iterations. If it is detected that the cost of *gbest* particle is not converging well, it will feedback to DIPA, which will do the necessary adjustment to the inheritance probabilities. It also samples the cost of the *gbest* particle in long iteration intervals and check whether the cost of *gbest* particle has decreased or not. If it is detected that the cost has not changed during the long interval, it will feedback to the DIPA that the algorithm is likely to have stuck in some local optima. In short, the convergence status as feedback by the feedback mechanism provides the information of the convergence rate and the indication of long period stuck at some local optima. DIPA will do the necessary adjustment when it receives the convergence status from the feedback mechanism. The detailed workings of DIPA can be left to the algorithm designer to decide.

Thus, the algorithm designer can just set an arbitrary set of the initial inheritance probabilities without worrying whether the set is a good set or bad set, since DIPA will adjust the probabilities dynamically in the algorithm.

3 Experimental Settings

Five numerical optimization experiments are conducted to compare the performance of PS-EA with GA and PSO in minimizing five popular test functions.

3.1 Test functions

The following five test functions are widely used for benchmarking purposes of optimization algorithms. For these functions, there are many local optima and/or saddles in their solution spaces. The amount of local optima and saddles increases with increasing complexity of the functions, i.e. with increasing dimension.

The first test function is Griewank function given by:

$$f_1 = \frac{1}{4000}\sum_{d=1}^{D} x_d^2 - \prod_{d=1}^{D}\cos\left(\frac{x_d}{\sqrt{d}}\right) + 1 \qquad (6)$$

The global minimum of f_1 is zero, and it occurs when x_d = 0 for all d = 1,2,… D. It has many widespread local minima. The locations of the minima are however regular distributed.

The second test function is Rastrigrin function given by:

$$f_2 = \sum_{d=1}^{D}\left(x_d^2 - 10\cos(2\pi x_d) + 10\right) \qquad (7)$$

The global minimum of f_2 is zero, and it occurs when x_d = 0 for all d = 1,2,… D. It is highly multimodel, and similar to Griewank, it has widespread local minima which are regularly distributed.

The third test function is Rosenbrock function given by:

$$f_3 = \sum_{d=1}^{D-1}\left(100\left(x_{d+1} - x_d^2\right)^2 + (x_d - 1)^2\right) \qquad (8)$$

The global minimum of f_3 is zero, and it occurs when x_d = 1 for all d = 1,2,… D. It is classic unimodel optimization problem. The global optimum is inside a long, narrow, parabolic shaped flat valley, popularly known as the Rosenbrock's valley. To find the valley is trivial, but to achieve convergence to the global optimum is difficult task.

The fourth test function is Ackley function given by:

$$f_4 = -20 + e - 20e^{\left(-0.2\sqrt{\frac{1}{D}\sum_{d=1}^{D} x_d^2}\right)} - e^{\frac{1}{D}\sum_{d=1}^{D}\cos(2\pi x_d)} \qquad (9)$$

The global minimum of f_4 is zero, and it occurs when x_d = 0 for all d = 1,2,… D. It is a highly multimodel function, similar to Griewank and Rastrigrin.

The fifth and last test function is Schwefel function given by:

$$f_5 = D \times 418.9829 - \sum_{d=1}^{D} x_d \sin\left(\sqrt{|x_d|}\right) \qquad (10)$$

The global minimum of f_5 is zero, and it occurs when x_d = 420.9867 for all d = 1,2,… D. Schwefel function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. The solution space is wickedly irregular with many local optima. The search algorithms are potentially prone to convergence in the wrong direction in the optimization of this function.

3.2 GA Scheme

The GA scheme used in the comparison experiment is used as suggested in a journal paper by Digalakis, J.G. and Margaritis, K.G. [2]. The scheme is shown in Table 1.

Table 1: GA Scheme

Crossover Probability	0.95
Crossover Scheme	Single point crossover
Crossover Parent Selection Scheme	Random selection (no elitism)
Mutation Probability	0.1
Child Production Scheme	A child chromosome is added to the population from any crossover or mutation operation.
New generation selection scheme	Best fixed number of chromosomes are selected from the "expanded" population of parent and child chromosomes for the next GA operations

3.4 PSO Scheme

The PSO scheme used in the experiment is used as suggested in a conference paper by M. Lvbjerg, T. K. Rasmussen and T. Krink [14]. The PSO equations used follows (1) and (2), except that (1) is modified to as follows:

$$v' = w.v + c1.U(0,1).(e1- x) + c2.U(0,1).(e2-x) \qquad (11)$$

where w is the additional initial weight, which varies from 0.9 to 0.7 linearly with the iterations.

The learning factors, c1 and c2 are set to be 2 and. The upper and lower bounds for v[], (V_{min}, V_{max}) are set to be the maximum upper and lower bounds of x, i.e. (V_{min}, V_{max}) = (*x_min*, *x_max*). If the sum of accelerations would cause the velocity on that dimension *v'*, to exceed V_{max} or V_{min}, , then the velocity on that dimension *v'*, is limited to V_{max} or V_{min} respectively.

3.3 PS-EA Scheme

An infeasible set of initial inheritance probabilities are used to test the performance of DIPA module of PS-EA. The purpose of setting the initial inheritance probabilities differently is to observe whether DIPA works properly to adjust the inheritance probabilities back to that ensures an effective search.

Having the inheritance probabilities to be set fixed as according Table 2 will result in slow convergence or even deterioration in the overall fitness of the swarm population. Thus, infeasible set of initial inheritance probabilities are purposely set to verify whether DIPA is able to adjust the set of inheritance probabilities for effective search.

3.4 Common experimental settings

The population size is set as 125 and the initialization ranges for the test functions are shown in Table 3.

Table 3: Initialization range for the test functions.

Function	Initialization range
f_1	$(-600,600)^n$
f_2	$(-15,15)^n$
f_3	$(-15,15)^n$
f_4	$(-32.768,32.768)^n$
f_5	$(-500,500)^n$

The corresponding maximum generation for each dimension of the test function is shown in Table 4.

Table 4: Maximum generation for each dimension of the test function.

Dimension	10	20	30
Maximum Generation	500	750	1000

Each algorithm will be run for each dimension (10, 20 and 30) for each test function for 30 trials and the mean best and standard deviation (unbiased) for runs will be collected for comparison.

4 Simulation Results and Discussion

After 30 trials of running each algorithm for each test function, the results on the best mean and the standard deviations were obtained and tabulated in Table 5. The best means obtained represents the performance of the algorithm in convergence, and the standard deviations obtained indicate the stability of the algorithms.

Table 5: Mean best costs and Standard deviation (unbiased) obtained from 30 trials for the optimization of each dimension

(a) ***Function dimension 10 and maximum generation of 500***

Func	GA		PSO		PS-EA	
	Mean best cost	Standard deviation	Mean best cost	Standard deviation	Mean best cost	Standard deviation
f_1	0.050228	0.029523	0.079393	0.033451	0.2223	0.0781
f_2	1.3928	0.76319	2.6559	1.3896	0.43404	0.2551
f_3	46.3184	33.8217	4.3713	2.3811	25.303	29.7964
f_4	0.59267	0.22482	9.8499×10^{-13}	9.6202×10^{-13}	0.19209	0.1951
f_5	1.9519	1.3044	161.87	144.16	0.32037	1.6185

(b) ***Function dimension 20 and maximum generation of 750***

Func	GA		PSO		PS-EA	
	Mean best cost	Standard deviation	Mean best cost	Standard deviation	Mean best cost	Standard deviation
f_1	1.0139	0.026966	0.030565	0.025419	0.59036	0.2030
f_2	6.0309	1.4537	12.059	3.3216	1.8135	0.2551
f_3	103.93	29.505	77.382	94.901	72.452	27.3441
f_4	0.92413	0.22599	1.1778×10^{-8}	1.5842×10^{-8}	0.32321	0.097353
f_5	7.285	2.9971	543.07	360.22	1.4984	0.84612

(c) ***Function dimension 30 and maximum generation 1000***

Func	GA		PSO		PS-EA	
	Mean best cost	Standard deviation	Mean best cost	Standard deviation	Mean best cost	Standard deviation
f_1	1.2342	0.11045	0.011151	0.014209	0.8211	0.1394
f_2	10.4388	2.6386	32.476	6.9521	3.0527	0.9985
f_3	166.283	59.5102	402.54	633.65	98.407	35.5791
f_4	1.0989	0.24956	1.4917×10^{-6}	1.8612×10^{-6}	0.3771	0.098762
f_5	13.5346	4.9534	990.77	581.14	3.272	1.6185

Table 5a shows the results of the simulation for the performance of each algorithm in optimizing the various functions of dimension 10 and maximum generation of 500 after running 30 trials. From the results in Table 5a, it is found that PS-EA outperforms GA and PSO in the optimization of Rastrigrin and Schwefel functions, which are highly multimodel, PS-EA has perform particularly well in the optimization of Schwefel, which is wickedly irregular with many local optima. PS-EA middle-performs in the optimization of Rosenbrock and Ackley functions, and worst-performs in the optimization of Griewank function. Despite the average performance PS-EA has displayed in optimizing Griewank, Rosenbrock and Ackley function, it can be explained on the fact that there is explicit probability in SUM that the particle can inherit parameter values of the not-so-good or even worse particles, like inheriting from the neighbor particles or undergo a random mutation in SUM. As compared to PSO or GA, which always lead the population with good solutions. The diversity factor introduced in PS-EA by SUM is believed to be the main reason for slow convergence in Griewank, Rosenbrock and Ackley function, but however, the results obtained by PS-EA are still within satisfactory region and it has shown its great search ability and local optima avoidance ability in its performance in Schwefel's function, which is well known for its deceptive property in its location of the global minimum, and to "cheat" optimization algorithms to converge in the wrong direction.

Table 5b shows the results of the simulation for the performance of each algorithm in optimizing the various functions of dimension 20 and maximum generation of 750 after running 30 trials. From the results in Table 5b, it is found that PS-EA outperforms GA and PSO in the optimization of Rastrigrin, Rosenbrock and Schwefel functions. It is noted for the optimization of Rosenbrock function, PS-EA has overtaken PSO as the best convergence performer at a higher dimension of 20 at just a margin. This suggests that PS-EA is performing better at a higher complexity. However, more conclusive result on the performance of PS-EA may be obtained when its performance maintains at higher function dimension of 30, which will be shown in the results in the next section. On the optimization of Griewank and Ackley functions, PS-EA remains as the middle performer, with PSO and GA as the best and worst performers respectively.

Table 5c shows the results of the simulation for the performance of each algorithm in optimizing the various functions of dimension 30 and maximum generation of 1000 after running 30 trials. From the results in Table 5c, it is found that PS-EA maintains itself as the best performer in the optimization of Rastrigrin, Rosenbrock and Schwefel functions, and a middle-performer in the optimization of Griewank and Ackley functions. For PS-EA performance in Rosenbrock function, PS-EA performs significantly better with a mean best cost of 98.407 versus 402.54 as obtained by PSO. Thus, it is conclusive to say that PS-EA is performing better in functions of high dimension and complexity.

With the results, PS-EA has proven experimentally to be a good optimization algorithm for multi-model test functions of Rastrigrin and Schwefel, as well as unimodel function of Rosenbrock. In the optimization of Griewank and Ackley functions, it middle-performs between GA and PSO. However, the results of PS-EA in the optimization of the two functions have been quite satisfactory, as the mean results are very near to the respective global minimum of the function. It is also observed that PS-EA is a good algorithm when the function dimension increases. It is shown in its optimization of Rastrigrin, Rosenbrock and Schwefel functions at the function dimension of 30 as in Table 5c, where the results obtained by PS-EA are much better than GA and PSO.

On the stability of PS-EA, except for Griewank and Ackley function, it has the best stability as compared to the PSO and GA. For Griewank, even though it is the least stable algorithm as compared to GA and PSO, the difference in the stability is considerably small. As for Ackley, the stability of PS-EA lies between GA and PSO. In overall, PS-EA is relative a stable algorithm that obtain reasonable consistent results.

5 Multiobjective optimization using PS-EA

In this section, we shall consider applying PS-EA to solve a classic multi-objective problem: Fonseca 2-objective minimization problem [15] as follows:

Minimize:

$$f_1(x_1, x_2, \ldots, x_8) = 1 - \exp\left(-\sum_{i=1}^{8} \left(x_i - \frac{1}{\sqrt{8}}\right)^2\right) \tag{11}$$

$$f_2(x_1, x_2, \ldots, x_8) = 1 - \exp\left(-\sum_{i=1}^{8} \left(x_i + \frac{1}{\sqrt{8}}\right)^2\right) \tag{12}$$

where $-4 \le x_i \le 4, \forall i = 1,2,\ldots,8$.

The true Pareto optimal set for this problem is $x_1 = x_2 = \ldots = x_8$, $-\frac{1}{\sqrt{8}} \le x_i \le \frac{1}{\sqrt{8}}, i = 1,2,\ldots,8$. To apply PS-EA in this multiobjective problem, the algorithm is modified as in Fig. 2.

1. Initialization of swarm
2. Evaluation of particles (based on shared fitness and Pareto ranking).
3. While (NOT maximum generation)
    a. Identify pbest
    b. PS-EA operations
    c. Evaluation of extended population
    d. Selection for next generation
4. Use either the final population of particles or pbest as the final solutions for the Pareto front.

Fig. 2. The flow of Multiobjective PS-EA (MOPS-EA).

With MOPS-EA as modified from PS-EA, simulation tests were conducted to solve the Fonseca's problem. A

sample objective curve, as shown in Fig. 3 is plotted at the

100th generation, showing the objective positions of *pbest* particles.

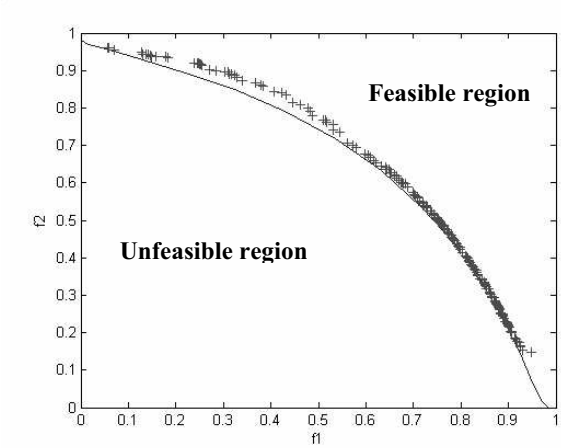

Fig. 3. Objective curve showing the true Pareto front and the found Pareto optimal set (+) at the 100th generation of PS-EA.

As shown in Fig. 3, it is observed that the final solution of *pbest* particles (as denoted by +) converges closely to the true Pareto front. This application establishes the applicability of this algorithm for searching in multi-modal, as well as, multiobjective problem domains.

6 Conclusion

PS-EA, as have been discussed, builds on the established works of EA and is extended with the workings of PSO. While retaining the flexibility to add heuristics into the algorithm, PS-EA does not compromise a great scale in its performance, as compared to original PSO. In this paper, PS-EA is compared to GA and PSO for optimization of five well-known test functions of Griewank, Rastrigrin, Rosenbrock, Ackley and Schwefel. PS-EA extends its capability as a potential multiobjective search algorithm in its application to minimize the Focenca 2-objective functions.

From the simulation results, some conclusive observations can be made as follows:

i. PS-EA is a good performer in optimizing difficult functions of high dimensions, like Rosenbrock, Rastrigrin and Schwefel functions.
ii. PS-EA is an average performer in optimizing fairly difficult functions like Griewank and Ackley functions.
iii. Even an infeasible set of initial inheritance probabilities will yield a good search by PS-EA. This confirms the operations of DIPA as a good dynamic parametric adjuster.
iv. The DIPA module of PS-EA has reduced the hassle of algorithm designers to set many parameters, and has worked well to adjust the inheritance probabilities.
v. PS-EA is well suited to for multi-modal functions of high dimension, as well as, multiojective problems.

Future works shall involve in applying PS-EA to solve real world problems, like scheduling, network routing and power forecasting. Future developments of MOPS-EA will also be continued.

References

[1] Anna Hondroudakis, Joel Malard and Gregory V. Wilson, **"**An Introduction to Genetic Algorithms Using RPL2: The EPIC Version", Computer Based Learning Unit, University of Leeds, 1995.

[2] Digalakis, J.G. and Margaritis, K.G., "An experimental study of benchmarking functions for genetic algorithms", 2000 IEEE International Conference on Systems, Man, and Cybernetics, pp. 3810 -3815 vol.5, 2000

[3] Sinclair, M.C, **"**The application of a genetic algorithm to trunk network routing table optimization", 10th.Performance Engineering in Telecommunications Network Teletraffic Symposium, pp. 2/1 -2/6, 1993.

[4] Greenwood, G.W.; Lang, C.; Hurley, S, "Scheduling tasks in real-time systems using evolutionary strategies", Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems, pp. 195-196, 1995.

[5] Fogel, D. and Sebald, A.V., "Use Of Evolutionary Programming In The Design Of Neural Networks For Artifact Detection", Proceedings of the Twelfth Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 1408 –1409, 1990.

[6] Meshref, H. and VanLandingham, H., "Artificial immune systems: application to autonomous agents", 2000 IEEE International Conference on Systems, Man, and Cybernetics, pp. 61 -66 vol.1, 2000.

[7] Calogero Di Stefano and Andrea G.B. Tettamanzi, "An Evolutionary Algorithm for Solving the School Time-Tabling Problem", Proceedings of Applications of Evolutionary Computing, pp. 452—462, 2001.

[8] Dipti Srinivasan, Tian Hou Seow and Jian Xin Xu, "Automated Time Table Generation Using Multiple Context for University Modules", Proceedings of IEEE Congress of Evolutionary Computation, Vol. 2, pp. 1751-1756, 2002.

[9] Dipti Srinivasan, Seow Tian Hou, Xu Jian Xin, "Constraint-Based University Time-Tabling Using Evolutionary Algorithm", Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning, Vol. 2, pp. 252-256, 2002.

[10] James Kennedy, Russell C. Eberhart, with Yuhui Shi, "Swarm Intelligence", Morgan Kaufman Publishers, 2002.

[11] Eberhart, Yuhui Shi, "Particle Swarm Optimization: developments, applications and resources", Proceedings of the 2001 Congress on Evolutionary Computation, , Vol 1, pp. 81-86, 2001.

[12] Chunkai Zhang; Huihe Shao; Yu Li, Systems, "Particle swarm optimisation for evolving artificial neural network", 2000 IEEE International Conference on Man, and Cybernetics,vol. 4, pp. 2487-2490 ,2000.

[13] Peter J. Angeline, "Using selection to improve particle swarm optimization", The 1998 IEEE International Conference on IEEE World Congress on Computational Intelligence Evolutionary Computation Proceedings, pp. 84 –89, 1998.

[14] M. Lvbjerg and T. Rasmussen and T. Krink, "Hybrid particle swarm optimiser with breeding and subpopulations", Proceedings of the Genetic and Evolutionary Computation Conference, 2001.

[15] K.C. Tan, T.H. Lee, E.F.Khor, "Evolutionary algorithms with goal and priority information for multiobjective optimzation," in Proc. 1999 Congr. Evolutionary Computation, vol. 1, Washington, DC, July 1999, pp. 106-113.